MIDI: Musical Instrument Digital Interface

By BOB MOOG

MIDI (Musical Instrument Digital Interface) is the specification for a set of digital codes for transmitting music control and timing information in real time, and for the hardware interface through which the codes are transmitted.

Basic ideas for a digital musical instrument interconnection standard (then called Universal Synthesizer Interface) were proposed by Dave Smith and Chet Wood at the 70th Convention of the Audio Engineering Society in November 1981. By the end of 1982, the name had been changed to Musical Instrument Digital Interface, and several electronic musical instrument manufacturers had agreed on a comprehensive technical specification. The first documented MIDI hookup between instruments of two different manufacturers took place, without formality or ceremony, at the National Association of Music Merchants convention in January of 1983.

In the three or so years since MIDI first saw the light of day as a functioning technical specification, millions of MIDI-equipped instruments have been made and sold, and countless thousands of music producers have embraced its capabilities. Onstage, electronic instrument performers use MIDI to network their synthesizers, sequencers, and rhythm machines. In the studio, personal computers equipped with sophisticated MIDI-based software are transforming the way music is composed electronically. In thousands of churches, living rooms, and garage studios, MIDI is enabling traditional and amateur musicians to produce high-quality music with modestly priced, readily available equipment. And, in academic laboratories and computer music facilities around the world, MIDI is providing an inexpensive link between large research computers and conveniently programmable, musician-oriented tone-producing instruments.

The meteoric (and largely unanticipated) rise in the popularity of MIDI, and in its importance to contemporary musicians, has created a blizzard of confusion and misconception among its users. Like all system specifications, MIDI was drawn up with certain goals, starting assumptions, and compromises in mind. Some of its limitations are easy to assess; others are still the subjects of discussion and disagreement, even among MIDI aficionados.

The purposes of this article are a) to explain what the MIDI specification is at the present time, and b) to review its applications—especially with respect to the operating characteristics of currently available equipment and software. We assume that the reader is familiar with synthesizers and basic audio studio techniques, and wants to use MIDI to get the most from his equipment. Discussions of the merits of the MIDI specification itself, or of the advisability of implementing alternate, higher performance interconnection standards, are primarily of academic interest, and are not included in this article.

Terms that have a specific, unique meaning in MIDI are shown in italics and are defined when they are first used in the text.

WHAT IS MIDI?

MIDI 1.0 DETAILED SPECIFICATION is the document that currently defines the Musical Instrument Digital Interface. The bulk of the specification describes the numerical codes that represent the various types of musical information that can be transmitted via MIDI. A description of the hardware that transmits and receives the MIDI signals occupies only two pages of the specification. We will now describe the hardware interface and mention some MIDI operating characteristics that are attributable to its hardware design.

THE MIDI HARDWARE INTERFACE

The MIDI hardware interface is a) digital, b) serial, and, to a limited extent, c) bidirectional. A *digital* interface transmits information as a stream of numbers. MIDI numbers are eight bits long, which means that each number specifies one of 256 possible values. A *serial* interface transmits numbers one bit at a time—a data transmission format that requires only a simple, two-wire cable. A *bidirectional* interface is capable of transmitting information in both directions. Because MIDI information flows entirely as a stream of numbers that must be formatted in the transmitting instrument and interpreted in the receiving instrument, a MIDI-equipped device must have a microprocessor-based digital operating system.

The schematic diagram of the standard MIDI hardware interface is shown in its entirety in Fig. 1. A MIDI instrument may have a MIDI IN, a MIDI THRU, and/or a MIDI OUT connector. MIDI IN receives MIDI data, MIDI THRU delivers an exact replica of the signal received at MIDI IN, and MIDI OUT delivers a signal that is generated or processed by the instrument itself. The connectors are standard 5-pin female DIN. The drive signal is a 5-mA current loop, which is set up when a standard MIDI cable is connected between one instrument's MIDI OUT (or MIDI THRU) and another's MIDI IN. The MIDI hardware interface connects to the instrument's microprocessor through a UART or similar logic circuit that is able to process serial data.

The MIDI cable consists of a shielded, twisted pair of wires and is terminated at both ends with five-pin male DIN plugs. In operation, only one end of the shield is grounded. This, and the electrical isolation afforded by the current loop, virtually eliminates contamination of an instrument's audio output with noise from the MIDI signal. MIDI cables may be up to 50 feet long.

MIDI serial data flows at the rate of 31.25 kilobits per second, a rate that is easily derived from common microprocessor clock rates (e.g., 1-megahertz clock, divided by 32). As we shall see, this rate of data flow does limit the amount of information that a single MIDI link can handle. However, a significantly faster data transmission rate would require a more expensive hardware interface, especially cables. Furthermore, most instruments' operating systems are barely able to keep up with the present MIDI data-transmission rate; increasing the MIDI transmission rate would not significantly decrease the overall system response time.

THE MIDI CODES

A MIDI data stream is organized into 10-bit words. The first bit is called the *start bit* (which is always 0), the next eight are the desired information, and the last is the *stop bit* (which is always 0). The start and stop bits delineate the desired data and do not carry information themselves. A complete word is transmitted in 320 microseconds, which is a about one third of a millisecond.

For the rest of this article, we shall concern ourselves with the values of the eight-bit bytes which fall between the start and stop bits. We shall write the bytes as strings of eight 1s and 0s, so it will be easy to visualize them as serial data. The leftmost digit is the first (lead) digit of the byte.

There are two main types of bytes: *status* and *data*. Data bytes tell the numerical value of a physical entity (e.g., the number of a key, or the position of a control on the panel), while a status byte serves as the address for the data bytes that follow it, or defines system states or events with which no numerical value is associated. The lead digit of a status byte is 1; that of a data byte is 0.

A MIDI *message* consists of a single status byte, immediately followed by 0, 1, or 2 data bytes. The MIDI specification gives the rules that govern how many data bytes are to be used to quantify a given status. (The exception is the so-called "system exclusive" status which allows the equipment manufacturer to specify how many data bytes are to be used.)

TYPES OF MIDI MESSAGES

Some MIDI messages are intended to be received by all instruments in the system, while others are confined to one of 16 *channels*. The channels are not separate hardware links (as they are in conventional analog audio systems), but rather are addresses to which some receiving instruments can be programmed to respond. As we shall see, the status byte of a *channel message* contains the four bits that indicate to which channel the message is addressed.

Messages that are addressed to the entire system are called *system messages*.

A status byte is of the form:

1aaabbbb.

The lead bit is always 1. The next three binary digits aaa indicate which type of status it is: If aaa = 111, then it is a system message type, and the binary digits bbbb specify the message in greater detail. Otherwise, it is a channel message type, and the binary digits bbbb indicate which channel.

All currently defined MIDI channel messages are listed below. Each message is described briefly. As we shall see, MIDI messages are generally couched in terms of piano-style keyboards, panel controls and switches, simple rhythmic structures, and other ubiquitous features of the contemporary music scene.

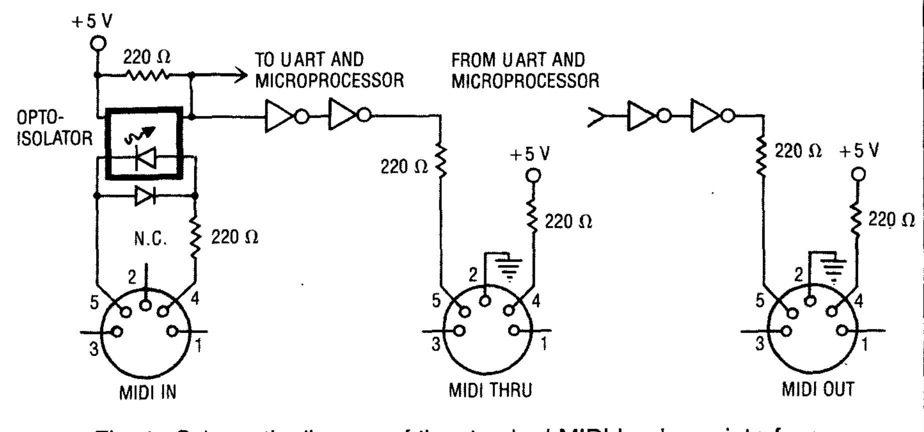


Fig. 1. Schematic diagram of the standard MIDI hardware interface.

CHANNEL MESSAGES

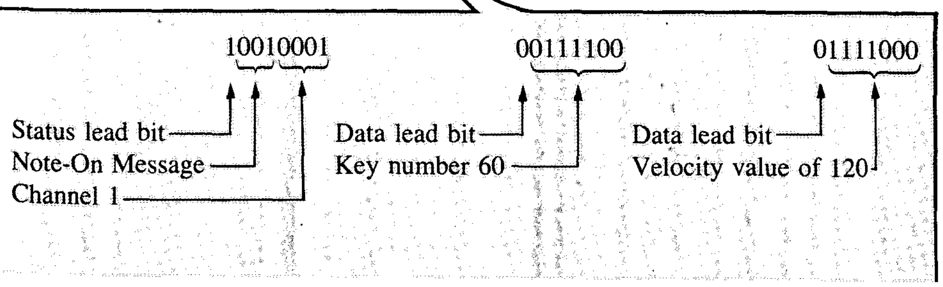
Note-On and Note-Off Events

A *Note-On Event* message is generated whenever a key of a piano-style music (as opposed to alphanumeric) keyboard is depressed. A *Note-Off Event* message is generated whenever a key a released.

Message	Status Byte	Data Byte(s)	
Note-Off Event	1000bbbb	0kkkkkkk	0vvvvvvv
Note-On Event	1001bbbb	0kkkkkkk	0vvvvvvv

In a Note-On Event, the seven data-carrying digits kkkkkkk of the first data byte specify which key was depressed, while the vvvvvvv of the second data byte specify the velocity at which the key descended. Both the key number and the velocity value may range from 0 to 127. Keys are numbered from bass to treble; middle C is numbered 60. The musical dynamic range from ppp (very soft) to fff (very loud), generally a difference of 20 to 30 dB, is covered by velocity values from 1 to 127, which correspond to key descent times of 40 milliseconds or so for a softly struck key to perhaps 5 milliseconds for a hard strike. The velocity value 0 means Note Off. MIDI keyboards that are not velocity sensitive are required to produce velocity values of 64. Key-on velocity information is typically used to determine a note's loudness, brightness, or attack time. In a Note-Off Event the vvvvvvv portion of the message tells how fast the key is released. The velocity range depends on the mechanical design of the keyboard. Only a few MIDI instruments currently produce, or respond to, key release velocity information. Uses of release velocity include control of note release time and release brightness.

For example, a MIDI message stream for middle C being struck hard, transmitted on channel 1, might be:



Polyphonic Key Pressure

Some keyboards are equipped to continuously detect the force that the player is exerting on each depressed (active) key. The format for transmitting *Polyphonic Key Pressure* message is:

Poly Key Pressure

1010bbbb 0kkkkkkk 0vvvvvvv

where the kkkkkkk of the first data byte is the key number, and the vvvvvvv of the second data byte is the value of force on that key at the time the message is transmitted.

Unlike key velocity information, which is transmitted once at the beginning and once at the end of an event, polyphonic key pressure information is a continuously variable parameter that must be transmitted many times per second. The MIDI specification does not tell the rate at which polyphonic key pressure is updated while the key is active. The manufacturer of the device sets this rate on the basis of such considerations as how fast the keyboard force sensors are scanned, how much microprocessor time is available, what the resolution of a sensor is, and how fast the force is changing.

Control Change

Aside from information from the keyboard, or from the Pitch Bend control device (to be discussed below), all commands to change operating parameters within the system are sent as *Control Change* messages. The format is:

Control Change

1011bbbb 0ccccccc 0vvvvvvv

(ccccccc ranges from 0 to 121)

where ccccccc of the first data byte is the numerical address of the control, and vvvvvvv is the new value of the control's output. The magnitude of ccccccc is restricted to the range 0 to 97. The numbers 98 to 121 are reserved for future expansion of the MIDI specification, while the numbers 122 to 127 denote operating-mode messages which will be described later.

In MIDI, the term *control* refers either to a continuous controller (such as a rotary potentiometer or one axis of a joystick) or to an off–on switch. Since vvvvvvv is a seven-bit number (one of 128 possible values), a single control change message transmits the control's setting with a resolution of one part in 128. In some applications (such as setting the amplitude of a tone within a mix), this resolution may be acceptable, while in other uses (such as the tuning of a pitched tone), the resolution must be much finer. For this reason, the Control Change numbers 0 to 31 are addresses for the most significant seven bits of 32 different continuous controllers, while the Control Change numbers 32 to 63 are addresses for the least significant seven bits of the same 32 continuous controllers. Thus, MIDI is able to transmit settings of continuous control devices to a resolution of 14 bits (one of 16,384 possible values). The manufacturer is free to decide if and when to specify the full 14 bits when transmitting Control Change information.

The Control Change numbers 64 to 97 are assigned to switches, although it is also permissible to use them for continuous controllers. When a Control Change message tells whether a switch is off or on, then vvvvvvv = 0 means that the switch is off, and vvvvvvv = 127 means that it is on. If a MIDI receiver is programmed to interpret a given Control Change address as a switch, then the switch is off when vvvvvvv is 0 to 63, and on when vvvvvvv is 64 to 127.

At the present time, there is no mandatory standard for assigning addresses to specific physical controls. However, MIDI acknowledges that some assignments are in common use. These are listed in Table 1.

Program Change

Whereas a control is linked to a single parameter of the sound, a program is a complete set of parameter values that, when taken together, define a voice or an instrument. The *Program Change* message format is:

Program Change

1100bbbb   0nnnnnnn.

Unlike the messages that we have discussed so far, Program Change has only one data byte. The number nnnnnnn is the address of the newly selected program, and may range from 0 to 127.

Channel Pressure and Pitch Bend Change

The Polyphonic Key Pressure message, described above, specifies the player's force on each active key. *Channel Pressure*, on the other hand, specifies a continuously varying control signal that is associated with a given channel. Occasionally, the word "aftertouch" is used instead of "pressure." The message format is:

Channel Pressure

1101bbbb   0vvvvvvv

where the data byte tells the parameter's new value.

Pitch Bend Change is another continuously varying control signal that is associated with a given channel. Its message format is:

Pitch Bend Change

1110bbbb 0vvvvvvv 0xxxxxxx.

The Channel Pressure signal generally comes from a keyboard that is equipped with only one sensor which detects the player's total force on all the keys. It is often used to provide rapid variation of a low-resolution global sound parameter such as brightness or loudness. The Pitch Bend signal generally comes from a wheel, joystick, or similar control device that can be manipulated rapidly and precisely, and, as its name indicates, is used to bend the pitch up or down from the nominal pitches determined by the active keys. The Channel Pressure message has only one data byte, which specifies the value to a 7-bit resolution. The Pitch Bend message, on the other hand, has two data bytes, which specify the value to 14 bits.

Channel Pressure and Pitch Bend messages have their own dedicated status codes, and therefore require fewer bytes per message and may be transmitted more rapidly than Control Change messages. Because of this, they are the preferred messages for conveying global, continuously varying performance parameters.

Channel Mode Messages

All messages described so far are referred to as *channel voice* messages. They specify the parameters that determine the operating points of the sound-producing hardware. We now list the *channel mode* messages which specify channel operating modes and functions that do not involve sound parameters.

Channel Mode messages are of the form:

Channel Mode

1011bbbb 0ccccccc 0vvvvvvv

(ccccccc ranges from 122 to 127)

The Channel Mode messages are summarized in Table 2.

The most important, and perhaps the most frequently misunderstood MIDI mode messages, are those having to do with voice assignment. A *voice* is a single sound-generating/shaping chain and its associated support circuitry and/or software. A typical keyboard synthesizer may have as few as one or as many as 16

Table 1. Commonly used control assignments.

Control number	Function
1	Modulation Wheel or Lever
2	Breath Controller
4	Foot Controller
5	Portamento Time
6	Data Entry
7	Main Volume
64	Damper (Sustain) Pedal
65	Portamento
66	Sostenuto
67	Soft Pedal
96	Data Increment
97	Data Decrement

Table 2. Channel Mode messages.

Message	Status byte	Data bytes	
Local Control Off	1011bbbb	01111010	00000000
Local Control On	1011bbbb	01111010	1111111
All Notes Off	1011bbbb	01111011	00000000
Omni Mode Off	1011bbbb	01111100	00000000
Omni Mode On	1011bbbb	01111101	00000000
Mono Mode On	1011bbbb	01111110	0zzzzzzz
Poly Mode On	1011bbbb	01111111	00000000

(Note: zzzzzzz = number of active channels) voices. The MIDI voice-assignment modes that determine to which channel messages a given voice responds are called Omni, Poly, and Mono.

A MIDI instrument that is in Omni mode receives Channel Voice messages without regard to the channel on which they are sent.

An instrument in Poly mode receives Channel Voice messages on only one channel, and routes the messages to all of its voices according to the instrument's voice assignment algorithm.

An instrument in Mono mode receives messages for only one voice per channel.

In terms of actual synthesizer hardware, the simplest mode to implement is Omni because channel numbers are simply ignored in this mode. Poly mode requires the ability to recognize and respond to channel number information. Instruments that are capable of operating in Mono mode are the most versatile of all, since they respond to a range of channel numbers. Since control change and program change commands are Channel Voice messages, a Mono mode instrument is *multitimbral*, which means that each of its voices may be programmed independently of the others.

As Table 2 shows, MIDI (by means of two messages) enables one to specify Omni mode to be either on or off in a synthesizer, while at the same time specifying either Poly or Mono mode. This gives four possible *mode combinations*, labeled 1 through 4. They are listed in Table 3.

Channel n is called the *basic channel*. Although MIDI has messages for switching a receiving instrument's mode, it has no messages for switching an instrument's basic channel. The user must set the basic channel of each instrument at the instrument's user interface. Not all MIDI instruments have means for changing their basic channels. The MIDI user must be aware of which basic channel(s) each of his instruments is able to transmit and receive on.

Generally, not all mode combinations are implemented in a given instrument. The current MIDI specification contains rules that determine what mode combination an instrument should default to if it receives a message to switch to an unimplemented mode.

Table 3. Mode combinations.

Number	Description
1	Omni On/Poly: Voice messages are received on all channels, but transmitted only on Channel n.
2	Omni On/Mono: Synthesizer receives Voice messages on all channels, but assigns them one at a time to control a single voice. Voice messages are transmitted on Channel n
3	Omni Off/Poly: Voice messages are both received and transmitted only on Channel n
4	Omni Off/Mono: Synthesizer receives Voice messages on a set of channels, beginning with Channel n, and assigns each active channel to one of its voices. Voice messages are transmitted on a set of channels, beginning with Channel n; messages are for only one voice per channel.

Running Status

Individual channel messages are two or three bytes long. It often happens that a long stream of messages of the same type is sent. For instance, when a synthesizer keyboard is being played, a frequently unbroken stream of Note-On messages is sent. Another example is pitch bend. A musician holding a chord while bending its pitch is generating an unbroken stream of Pitch Bend messages. If individual messages are sent to convey these types of information, as much as 50% of the message transmission time is taken up with redundant status bytes.

To save transmission time when a stream of same-status messages is being sent, MIDI has a data transmission mode, called *running status*, to which MIDI receivers are required to respond. Once a status byte for a channel message is received, the receiving instrument must remember that status, and interpret all data subsequently received as also belonging to that status, until a new status byte is received. Thus, in Running Status mode, a stream of Note-On events (remember, a Note-Off event can be formatted as a Note-On event with zero velocity) takes two bytes per event to transmit (instead of three, for nonrunning-status), and a stream of Channel Pressure updates takes one byte per update (instead of two).

SYSTEM MESSAGES

System messages are intended for all devices in a MIDI system, and therefore do not carry channel labels. There are three types of system messages: *Real Time*, *Common*, and *Exclusive*. System messages always start with a status byte whose first four bits are 1111.

System Real-Time Messages

Just as Channel messages imply a set of programmable voices, Real-Time system messages imply the presence of one or more sequencers (real-time digital data recorders). All Real-Time messages are just one byte long and are used for synchronizing the entire system in real time. The Real-Time messages are:

Timing Clock	11111000
Start	11111010
Continue	11111011
Stop	11111100
Active Sensing	11111110
System Reset	11111111

The Timing Clock is transmitted continuously, 24 to the quarter note. Start activates a recorded sequence from the beginning, Stop stops the sequence, and Continue restarts the sequence without resetting to the beginning. Active Sensing, when used, is transmitted approximately three times per second to indicate that the transmitting device is active. System Reset initializes the entire system, as if the power had just been turned on.

System Common Messages

There are four *System Common* messages. Two are used to access specific locations in sequencer memory. They are:

Song Position Pointer

11110010 0lllllll 0hhhhhhh

Song Select

11110011 0sssssss

Song Position Pointer enables the user to start or continue a sequence at a specific point. hhhhhhh and lllllll are the most significant and least significant bits, respectively, of a 14-bit number which is the number of beats (6 MIDI clocks = 1 beat) from the beginning of the song. *Song Select* enables the user to select one of 128 song files within the sequencer library memory. The data byte 0sssssss tells the number of the song.

The other two System Common messages are:

Tune Request	11110110
End System Exclusive	11110111

Tune Request simply initiates routines within analog synthesizers to tune the oscillators. *End System Exclusive* is a flag that signals the end of a System Exclusive message.

System Exclusive Messages

Registered equipment manufacturers may use a *System Exclusive* message to transfer special, instrument-specific information. The System Exclusive message is of the following form:

System Exclusive

```
11110000
    0iiiiiii
    0xxxxxxx
    :
    0xxxxxxx
    11110111
```

The first byte is the System Exclusive status byte. The second byte is the manufacturer's numerical identification assigned to registered manufacturers either by the MIDI Manufacturer's Association or by the Japan MIDI Standards Association. An arbitrary number of data bytes 0xxxxxxx carry the desired information. The last byte is the End of Exclusive (System Common) message. System Exclusive messages may not be interleaved with any other messages, except Real-Time messages.

We have now covered the basic features of the MIDI codeset (language) and of the MIDI hardware interface. Our discussion has been in terms of the bits making up the actual status and data bytes of MIDI message streams. While the simpler uses of MIDI do not involve the user in the details of the message structures, MIDI is versatile and open-ended enough to encourage the creative and adventurous user to construct and use systems of considerable complexity with programmable MIDI controllers and processors. Just as one has to understand the details of how analog audio instruments work in order to get the most performance out of one's equipment, so must one understand MIDI down to the 'bit level' in order to take full advantage of its capabilities.

We now discuss some considerations that must be taken into account when using MIDI.

MIDI NETWORKS

In the MIDI network diagrams that follow, an instrument is represented by a box with three compartments I (MIDI IN), O (MIDI OUT), and T (MIDI THRU), as well as A (audio output).

The simplest MIDI network (Fig. 2) is a master-slave setup, in which the master (typically a MIDI keyboard or a keyboard-controlled synthesizer) 'plays' one or more satellite sound-producing instruments. Historically speaking, this is the first way that MIDI was used, and is still important to synthesists who wish to build interesting, complex tones by combining sonic resources of several synthesizers. The data density generated by one musician playing a conventional MIDI keyboard is not high enough to cause problems or require special consideration. Networks of this type, in which the MIDI signal is routed from MIDI OUT (or MIDI THRU) of one device, to MIDI IN of the next device, are called *chain networks*.

A step up in complexity (Fig. 3) is a typical drum-machine-plus-synthesizer network, where the MIDI clock from the drum machine drives a sequencer in the synthesizer. The synthesizer now receives MIDI Real-Time messages and must process these along with its own keyboard information. Since all information going in or out of the synthesizer must be handled sequentially by the instrument's microprocessor, it is not unlikely that some synthesizers in a network like this will lag enough to produce an audible delay.

A simple bidirectional network (Fig. 4) uses an external sequencer to store performance data from the synthesizer and play it back on command. MIDI controllers such as stand-alone sequencers generally produce message streams of a higher density than those which are generated by a single synthesizer being performed in real time. The satellite synthesizers on the chain may be set to receive on different channels, so the sequencer can be used to orchestrate each synthesizer independently of the others. Although voice messages to the satellites go out on different channels, they are shown here going through a single cable chain, and therefore must go sequentially. For this reason, chain networks are susceptible to audible delays.

A typical large network (Fig. 5) might use a personal computer equipped with a MIDI interface to communicate with a multiplicity of synthesizers and drum machines. Currently available computer-MIDI interfaces have two to eight MIDI OUT ports. This allows the computer to communicate with two or more instruments simultaneously instead of sequentially, thereby greatly reducing delays associated wtih sending data on a multiplicity of channels through a single cable. Networks centered around a MIDI controller with more than one set of ports are called *star networks*.

Of course, a MIDI system may have both chain and star portions. The shorter the chains, the less likely it is that audible delays will be large enough to be a problem. The number of possible star branches is determined by the number of hardware ports on the system master controller. MIDI does not specify how many ports an instrument may have; that decision is made by the manufacturer.

Currently available MIDI computer software packages support as many as eight hardware ports. The more advanced sequencer programs support a total of 32 channels. Since one port can send 16 MIDI channels at most, a 32-channel system is actually two separate MIDI subsystems that are tied together by the software. The master controller for the system must have at least two hardware ports.

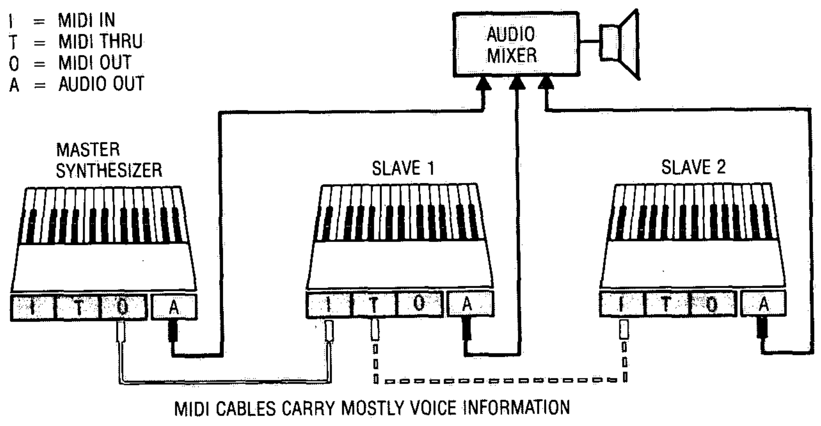


Fig. 2. Master-slave MIDI network.

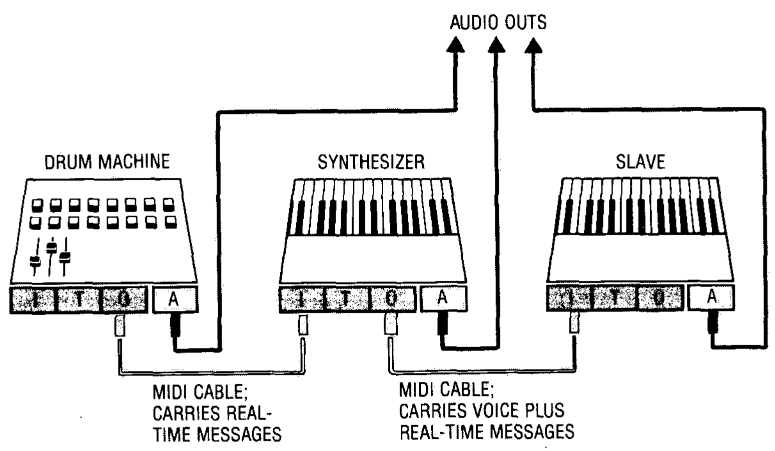

AUDIO OUTS
DRUM MACHINE
SYNTHESIZER
SLAVE
I
T
O
A
MIDI CABLE; CARRIES REAL-TIME MESSAGES
MIDI CABLE; CARRIES VOICE PLUS REAL-TIME MESSAGES

Fig. 3. MIDI network synchronized to drum machine.

DELAYS IN MIDI SYSTEMS

A functioning MIDI system has several sources of message-processing and transmission delay. Some are inherent in MIDI itself, while others are limitations of specific instruments. Starting, say, with a master synthesizer, there is a delay due to the processing time required to analyze keyboard signals, calculate velocities, and format the information into a MIDI message. Next is the time taken to actually send the messages. For instance, transmitting the messages to turn on a full, six-note chord takes four to six milliseconds. Third are delays due to message contention. MIDI clocks, Note-On and Note-Off information, and control-change message may arrive at the same time, and would have to be assigned priorities and then sent sequentially. And finally, decoding and implementing MIDI messages at the receiving synthesizer also require microprocessor time. The total system delay, even in the simplest link, is seldom less than 10 milliseconds, and can be as much as 20 or 30 milliseconds.

How much delay is too much? That depends on what you are trying to synchronize. For instance, it is not possible to use MIDI to connect two synthesizers when their audio waveforms must have a specified phase difference. In this application, the time difference between the two waveforms must be accurate to within a small fraction of a millisecond, which is beyond the time resolution of MIDI. If you are enhancing one synthesizer's sound by adding a sharp transient attack from another synthesizer, or if you are constructing precise rhythm patterns from sounds made in different instruments, then the delays must be controlled to within a very few milliseconds. If, on the other hand, you are combining slow-attack sounds, time differences of 20 or 30 milliseconds are generally acceptable.

The easiest way of managing time delays in MIDI systems is to reduce the number of channels sent through any one MIDI cable. One channel per MIDI cable is the ideal. This requires the use of a star network with as many MIDI OUTs on the controller as there are receiving instruments in the system. Another way of managing time delays, especially during playback of a complex sequence that has already been assembled, is to program in precise delays at the master controller, on an instrument-by-instrument basis, so that all instruments are closely synchronized to the slowest instrument.

SYNCHRONIZING MIDI SYSTEMS WITH SMPTE

Sequencers preserve the temporal arrangement of a MIDI message stream by storing the time at which a message was received, at the same time the message itself is stored. If these *time tags* are recorded as divisions of the MIDI Timing Clock intervals, then the location of a specific point in time within the message stream can be determined to within a few milliseconds, merely by specifying the song pointer position immediately before the desired point and the value of the time tag at that point.

Instruments that translate SMPTE time code into corresponding MIDI Song Position Pointer and Real-Time messages are called *SMPTE-to-MIDI converters*, and are widely used in applications where MIDI systems must be synchronized to multitrack tape. Unlike the SMPTE code signal, whose bandwidth easily falls within the audio range, the MIDI message stream cannot be recorded by conventional audio equipment.

MIDI DATA RESOLUTION

MIDI Control-Change messages may be sent with 7-bit (1 part in 128) or 14-bit (1 part in 16K) resolution. Poly Key Pressure and Channel Pressure are sent with 7-bit resolution, while Pitch Bend is sent with 14-bit resolution.

The MIDI specification says nothing about how much these control signals change the corresponding sound parameters, or, for that matter, what the corresponding sound parameters are.

Pitch Bend is perhaps the most critical continuous MIDI control signal, because our ears are extremely sensitive to slight pitch errors. The total range of pitch bending is set at the receiving instrument by the user. For control from a conventional pitch bend lever, popular range values are ± 2 semitones to ± 1 octave. For the octave range, a 14-bit resolution gives a minimum pitch step of much less than 1 cent (1 cent = 1/100 semitone), which is negligible. For control from a pitch tracker or similar

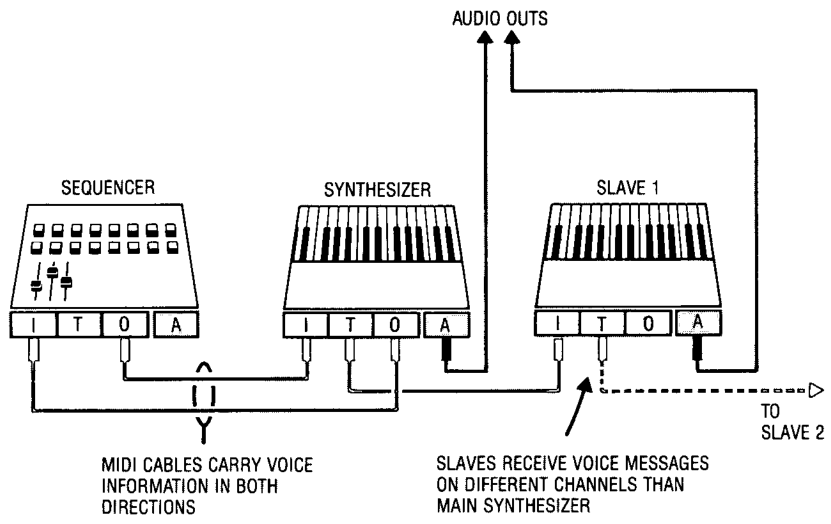

AUDIO OUTS
SEQUENCER
SYNTHESIZER
SLAVE 1
I T O A
I T O A
I T O A
TO SLAVE 2
MIDI CABLES CARRY VOICE INFORMATION IN BOTH DIRECTIONS
SLAVES RECEIVE VOICE MESSAGES ON DIFFERENT CHANNELS THAN MAIN SYNTHESIZER

Fig. 4. Sequencer-controlled MIDI network with one or more satellite slaves.

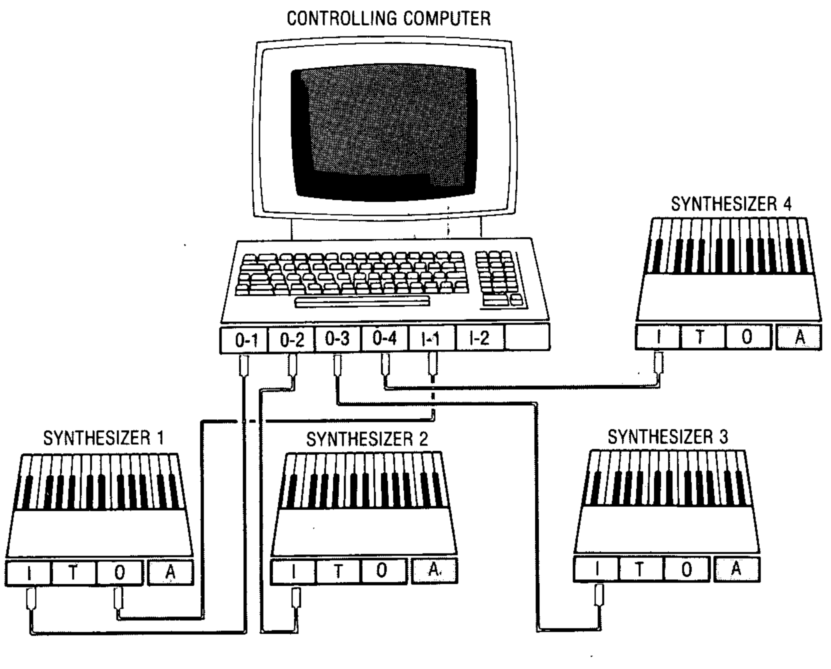
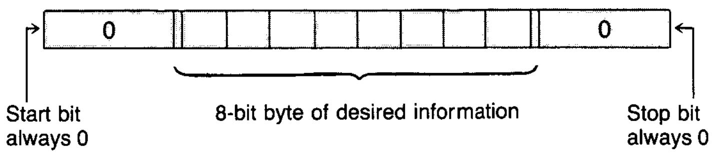


Fig. 5. Star network of four synthesizers and a personal computer. Computer MIDI interface has four MIDI outs and two MIDI ins.

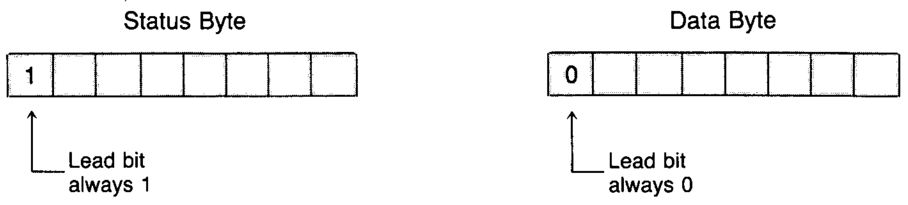
SUMMARY OF THE MIDI CODES

MIDI information is transmitted serially as a stream of 10-bit digital words. Each word is transmitted in 320 microseconds, and consists of a start bit, an eight-bit byte of desired information, and a stop bit.

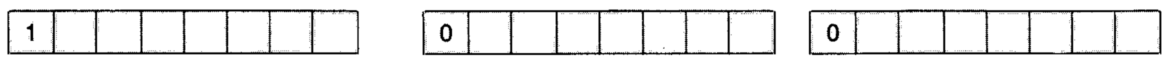


Start and stop bits are omitted from the rest of this summary.

There are two types of bytes: Status and Data. Status bytes tell the meaning of the data bytes that follow, or define system states or events with no numerical value. Data bytes give numerical values of physical entities, or numeric representations of specific messages.



A MIDI Message generally consists of one status byte, followed by 0, 1, or 2 data bytes.



A status byte is of the form:

1 a a a b b b b

If aaa is not 111, the message is called a Channel Message. The message is for one of the 16 MIDI channels; bbbb specifies which channel. If aaa is 111, the message is called a System Message. The message is for the entire system; bbbb specifies the message.

CURRENTLY DEFINED MIDI MESSAGES

CHANNEL MESSAGES

Channel Voice Messages specify parameters which determine the operating points of the sound-producing instruments.

MESSAGE	STATUS BYTE	DATA BYTES	
Note-on Event	$\underbrace{1001}_{\uparrow}$bbbb Note-on Event	0$\underbrace{kkkkkkk}_{\uparrow}$ Key number (0-127)	0$\underbrace{vvvvvvv}_{\uparrow}$ Key-down velocity (0 = note off)
Note-off Event	$\underbrace{1000}_{\uparrow}$bbbb Note-off Event	0$\underbrace{kkkkkkk}_{\uparrow}$ Key number (0-127)	0$\underbrace{vvvvvvv}_{\uparrow}$ Key-up velocity
Polyphonic Key Pressure	$\underbrace{1010}_{\uparrow}$bbbb Poly Key Pressure	0$\underbrace{kkkkkkk}_{\uparrow}$ Key number (0-127)	0$\underbrace{vvvvvvv}_{\uparrow}$ Force on the key
Control Change	$\underbrace{1011}_{\uparrow}$bbbb Control Change	0$\underbrace{ccccccc}_{\uparrow}$ Address of control (0-121)	0$\underbrace{vvvvvvv}_{\uparrow}$ Value of control's output

MESSAGE	STATUS BYTE	DATA BYTES
Program Change	$\underbrace{1100}_{\uparrow\ \text{Program Change}}\text{bbbb}$	$\underbrace{0\text{nnnnnnn}}_{\uparrow\ \text{Number of newly selected program}}$
Channel Pressure	$\underbrace{1101}_{\uparrow\ \text{Channel Pressure}}\text{bbbb}$	$\underbrace{0\text{vvvvvvv}}_{\uparrow\ \text{Value of continuously variable control with 7-bit resolution}}$
Pitch Bend	$\underbrace{1110}_{\uparrow\ \text{Pitch Bend}}\text{bbbb}$	$\underbrace{0\text{lllllll} \qquad 0\text{hhhhhhh}}_{\text{Value of continuously variable control with 14-bit resolution}}$

Channel Mode Messages specify channel operating modes, and functions not involving sound parameters. In all channel mode messages, the status byte is of the form 1011bbbb, and the first data byte ranges from 122 to 127.

Local Control Off	1011bbbb	01111010	0000000
Local Control On	1011bbbb	01111010	1111111
All Notes Off	1011bbbb	01111011	0000000
Omni Mode Off	1011bbbb	01111100	0000000
Omni Mode On	1011bbbb	01111101	0000000
Mono Mode On	1011bbbb	01111110	0zzzzzz
Poly Mode On	1011bbbb	01111111	0000000

(Note: zzzzzzz = number of channels)

SYSTEM MESSAGES

System Real-Time Messages synchronize timekeeping functions in the entire system. No data bytes are used. All System Real-Time messages are of the form 11111xxx.

Timing Clock	11111000
Start	11111010
Continue	11111011
Stop	11111100
Active Sensing	11111110
System Reset	11111111

System Common Messages provide non-real-time system information. All System Common status bytes are of the form 11110xxx.

Song Postion Pointer	11110010	$\underbrace{0\text{lllllll} \qquad 0\text{hhhhhhh}}_{\text{14-bit designation of number of beats from beginning of song.}}$
Song Select	11110011	$\underbrace{0\text{sssssss}}_{\uparrow\ \text{Number of song}}$
Tune Request	11110110	
End System Exclusive	11110111	

System Exclusive Messages are defined by the equipment manufacturer and may be of any length.

System Exclusive	11110000	(Begin System Exclusive Message)
	0iiiiiii	(Manufacturer's identification number)
	$\left.\begin{array}{c}0\text{xxxxxxx}\\ \cdot\\ \cdot\\ \cdot\\ 0\text{xxxxxxx}\end{array}\right\}$	(Body of System Exclusive Message)
	11110111	(End System Exclusive Message)

device that may be called upon to send large continuous pitch changes, Pitch Bend may be used to cover an instrument's entire range with no audible pitch granularity.

Other continuously variable synthesizer parameters that require higher-than-7-bit resolution are: frequency ratios between oscillators, filter cutoff, and envelope times. Parameters that generally require only 7-bit resolution are: audio mix levels, waveform width, and FM modulation index. On currently available instruments, Control-Change messages are generally sent with 7-bit resolution.

DATA UPDATE RATES

In transmitting continuously variable control changes via MIDI, the rate at which values are updated are as important as the accuracy of the values themselves. When a panel control is used to trim up or slowly change a parameter, the update rate can be as slow as a few times a second. However, for player controls, such as Pitch Bend and Breath Controller, update rates on the order of 100 to 200 per second are necessary to reduce the update steps to inaudibility. Control update rates are set by the manufacturer.

In dense MIDI message streams, Control-Change messages are often given low priority and are transmitted significantly later than they were generated. If several controller update streams are being sent simultaneously (for instance, polyphonic key pressure from several keys, Breath Controller, and foot pedal), the data transmission may lag audibly or may be deliberately degraded by the instrument's operating algorithms. The system user must be aware that the use of a multiplicity of continuous controllers may easily saturate a MIDI link.

MIDI CONTROLLERS

Much interest exists in the development of new control interfaces with MIDI outputs. Several manufacturers now offer stand-alone keyboard controllers that combine sophisticated MIDI programming systems with novel keyboard touch-sensitivity schemes and auxiliary control devices. Some feature multiple-touch sensitivity (two or more continuously variable outputs *from each key*), while others have a multiplicity of MIDI OUTs.

Pitch-to-MIDI converters with sophisticated, real-time digital signal processing that rapidly and accurately derives the pitch of an audio tone, are in wide use by non-keyboard musicians. Pitch trackers for guitar, traditional orchestral instruments, and voice, are all currently available.

The rise in popularity of non-keyboard MIDI controllers, and new keyboard designs that offer expanded opportunities for musicians to impart continuous control over many sound parameters, will spur the development of sound-producing MIDI instruments that are designed to respond to several continuous control signals simultaneously. Designing MIDI networks to handle the increased data flow required by these new instruments will remain a challenge for the forseeable future.

FOR FURTHER INFORMATION

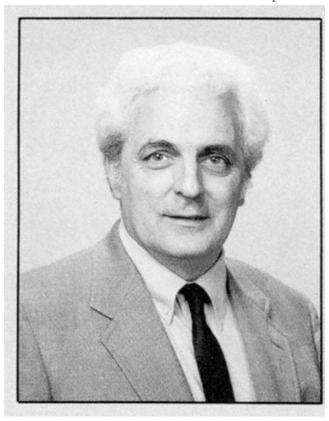
As the use of MIDI continues to grow, and new equipment and techniques are developed, musicians and studio personnel will want to study some of MIDI's more advanced features and capabilities. Some publications of interest are:

MIDI 1.0 Detailed Specification is the current official document of the MIDI Manufacturer's Association. It may be obtained from the International MIDI Association, 11857 Hartsook Street, North Hollywood, CA 91607. Telephone (818) 505-8964. The IMA also publishes a monthly newsletter.

MIDI for Musicians, by Craig Anderton. Amsco Publications, New York (1986). An extensive, nontechnical, introductory-level discussion of MIDI and its uses.

KEYBOARD Magazine, 1986 January. This special issue on MIDI contains a wide range of articles on specific MIDI topics.

"Musicians Make a Standard: The MIDI Phenomenon," by Gareth Loy. *Computer Music Journal*, vol. 9, no. 4, 1985 Winter. Discusses MIDI from an academic, computer-musician's perspective.



THE AUTHOR

Robert A. Moog received a B.S. in physics from Queens College (1957), a B.S. in electrical engineering from Columbia University (1957), and a Ph.D. in engineering physics from Cornell University (1965).

He has been actively engaged in the design of electronic music instruments since 1954 when he began the R. A. Moog Co. as a part-time business. It became a full-time operation in 1964 at which time electronic music synthesizer components were introduced. The company was incorporated in 1968, its name was changed in 1971 to Moog Music Inc., and, in 1973, it became a division of Norlin Industries. Dr. Moog remained as president of Moog Music until the end of 1977. In 1978 he formed Big Briar, Inc. for the purpose of designing and manufacturing custom electronic music equipment.

In 1984, Dr. Moog joined Kurzweil Music Systems, Waltham, MA, where he is currently vice president of new product research.